T.Y. Diploma : Sem. VI
[CO/CD/CM]

Software Testing

Time: 3 Hrs.] Prelim Question Paper Solution **[Marks : 100**

Q.1(a) Attempt any THREE of the following: [12]

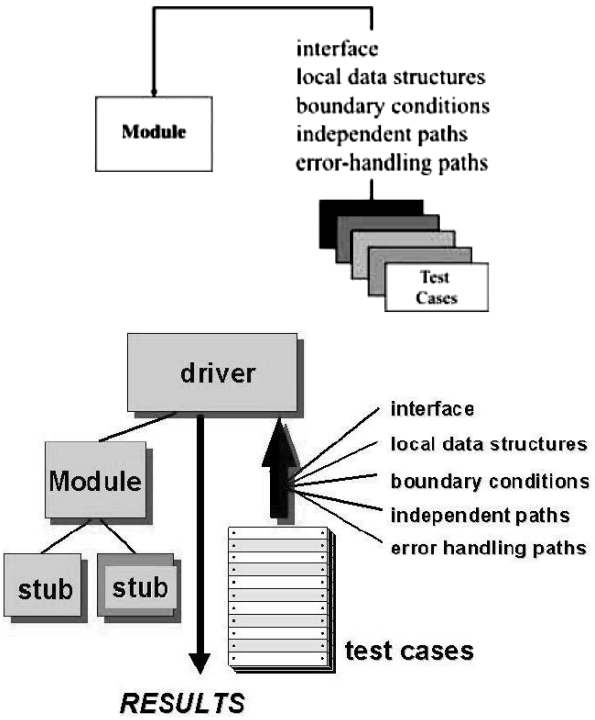
Q.1(a) (i) With the help of neat diagram, describe unit testing. [4]

Ans.: Unit Testing: Software product is made up of many units, each unit needed to be tested to find whether they have implemented the design correctly or not.

Additional Requirements: The module under consideration might be getting inputs from another module or the module is calling some another module. Some interface modules has to be simulated if required like drivers and stubs.

Drivers: The module where the required inputs for the module under test are simulated for the purpose of module or unit testing is known as a Driver module. The driver module may print or interpret the result produced by the module under test.

Stubs: The module under testing may also call some other module which is not ready at the time of testing. There is need of dummy modules required to simulate for testing, instead of actual modules. These are called stubs.



Unit testing procedure:

Unit testing is normally considered as an adjunct to the coding step. The design of unit test can be performed before coding begins or after source code has been generated. Guidance for establishing test cases for finding out undiscovered errors can be taken by the review of design information. Each test case need to be associated with set of expected results with it. In many applications a driver is known as "main program" that takes input from test case data, also passes that data to the component that need to be tested and prints concerned results.

Stubs are useful for replacing modules which are subordinate of the component that we are going to test. A dummy sub program or stub can make use of subordinate modules interface. It generally does very less data manipulation, also provides verification of entry and used to return control to the module i.e. currently undergoing the testing.

Q.1(a) (ii) What is Software testing? State objectives of Software testing. [4]

Ans.: **Software testing:** Software testing is defined as performing Verification and Validation of the Software Product for its correctness and accuracy of working. Software Testing is the process of executing a program with the intent of finding errors. A successful test is one that uncovers an as-yet-undiscovered error. Testing can show the presence of bugs but never their absence. Testing is a support function that helps developers look good by finding their mistakes before anyone else does. Execution of a work product with intent to find a defect.

Objectives of software testing are as follows:

1. Finding defects which may get created by a programmer while developing the software.
2. Gaining confidence in and providing information about the level of quality.
3. To prevent defects.
4. To make sure that the end result meets the business and user requirements.
5. To ensure that it satisfies the BRS that is Business Requirement Specification and SRS that is System Requirement Specifications.
6. To gain the confidence of the customers by providing them a quality product.

Q.1(a) (iii) What is Black Box testing? List any four techniques of Black Box testing. [4]

Ans.: Black Box testing involves looking at the specifications and does not require examining the code of the program. It is done from customer's point of view. The testers know the input and expected output. They will check whether with given input they are getting expected output or not.

Different techniques of Black Box test are:

1. Requirement base testing
2. Positive negative testing
3. Boundary value analysis
4. Decision tables
5. Equivalence partitioning
6. State based testing
7. Compatibility testing
8. User documentation testing
9. Domain testing

Q.1(a) (iv) Describe Inspection under static testing. [4]

Ans.: Under Static testing is to review the code without executing it. Inspection is the most formal method in static testing. This method can detect all faults, violations and other side effects.

OR

Inspection is formal review where people external to the testing team may be involved as inspectors. They are subject matter experts who review the work product.
In this:

1. Thorough preparation is required before an inspection/review
2. Enlisting multiple diverse views.
3. Assigning specific roles to the multiple participants
4. Going sequentially through the code in a structured manner.

There are four roles in inspection:

1. **Author of the code:** the person who had written the code
2. **Moderator:** who is expected to formally run the inspection according to the process?
3. **Inspectors:** are the people who actually provide review comments for the code.
4. **Scribe:** who takes detail notes during the inspection meeting and circulates them to the inspection team after the meeting.

The author or moderator selects review team. The inspection team assembles at the agreed

time for inspection meeting. The moderator takes the team sequentially through the program code. If any defect is found they will classify it as minor or major. A scribe documents the defects. For major defects the review team meets again to check whether the bugs are resolved or not.

Q.1(b) Attempt any ONE of the following: **[6]**

Q.1(b) (i) Describe Defect Management Process with neat & labelled diagram. **[6]**

Ans.: Defect management process diagram:

As shown in the above diagram the defect management process is divided into following tasks:

(i) Defect Prevention: Implementation of techniques, methodology and standard processes to reduce the risk of defects.

(ii) Deliverable Baseline: Establishment of milestones where deliverables will be considered complete and ready for further development work. When a deliverable is base lined, any further changes are controlled. Errors in a deliverable are not considered defects until after the deliverable is base lined.

(iii) Defect Discovery: Identification and reporting of defects for development team acknowledgment. A defect is only termed discovered when it has been documented and acknowledged as a valid defect by the development team member(s) responsible for the component(s) in error.

(iv) Defect Resolution: Work by the development team to prioritize, schedule and fix a defect, and document the resolution. This also includes notification back to the tester to ensure that the resolution is verified.

Q.1(b) (ii) What are the points considered while estimating impact of a defect? Also explain techniques to find defect. **[6]**

Ans.: Estimate Expected Impact of a Defect, Techniques for Finding Defects, Reporting a Defect. Once the critical risks are identified, the financial impact of each risk should be estimated. This can be done by assessing the impact, in dollars, if the risk does become a problem combined with the probability that the risk will become a problem. The product of these two numbers is the expected impact of the risk. The expected impact of a risk (E) is calculated as E = P * I, where: P = probability of the risk becoming a problem and I = Impact in dollars if the risk becomes a problem. Once the expected impact of each risk is identified, the risks should be prioritized by the expected impact and the degree to which the expected impact can be reduced. While guess work will constitute a major role in producing these numbers, precision is not important. What will be important is to identify the risk, and determine the risk's order of magnitude. Large, complex systems will have many critical risks. Whatever can be done to reduce the probability of each individual critical risk becoming a problem to a very small number should be done. Doing this increases the probability of a successful project by increasing the probability that none of the critical risks will become a problem.

Example:

- An organization with a project of 2,500 function points and was about medium at defect discovery and removal would have 1,650 defects remaining after all defect removal and discovery activities.

- The calculation is 2,500 × 1.2 = 3,000 potential defects.
- The organization would be able to remove about 45% of the defects or 1,350 defects.
- The total potential defects (3,000) less the removed defects (1,350) equals the remaining defects of 1,650.

Estimate Expected Impact of a Defect:

(i) There is a strong relationship between the number of test cases and the number of function points.
(ii) There is a strong relationship between the number of defects and the number of test cases and number of function points.
(iii) The number of acceptance test cases can be estimated by multiplying the number of function points by 1.2.
(iv) Acceptance test cases should be independent of technology and implementation techniques.
(v) If a software project was 100 function points the estimated number of test cases would be 120.
(vi) To estimate the number of potential defects is more involved.

Techniques to find defects:

(a) Quick Attacks
(b) Equivalence and Boundary Conditions
(c) Common Failure Modes
(d) State-Transition Diagrams
(e) Use Cases
(f) Code-Based Coverage Models
(g) Regression and High-Volume Test Techniques

{**Note: Following explanation is optional**}

(a) Quick Attacks:

- The quick-attacks technique allows you to perform a cursory analysis of a system in a very compressed timeframe.
- Even without a specification, you know a little bit about the software, so the time spent is also time invested in developing expertise.

(b) Equivalence and Boundary Conditions:

- Boundaries and equivalence classes give us a technique to reduce an infinite test set into something manageable.
- They also provide a mechanism for us to show that the requirements are "covered".

(c) Common Failure Modes:

- The heart of this method is to figure out what failures are common for the platform, the project, or the team; then try that test again on this build.
- If your team is new, or you haven't previously tracked bugs, you can still write down defects that "feel" recurring as they occur—and start checking for them.
- The more your team stretches itself (using a new database, new programming language, new team members, etc.), riskier the project will be—and, at the same time, the less valuable this technique will be.

(d) State-Transition Diagrams:

- Mapping out the application provides a list of immediate, powerful test ideas.
- Model can be improved by collaborating with the whole team to find "hidden" states—transitions that might be known only by the original programmer or specification author.
- Once you have the map, you can have other people draw their own diagrams, and then compare theirs to yours.
- The differences in those maps can indicate gaps in the requirements, defects in the

software, or at least different expectations among team members.

- The map you draw doesn't actually reflect how the software will operate; in other words, "the map is not the territory."
- Drawing a diagram won't find these differences,
- Like just about every other technique on this list, a state-transition diagram can be helpful, but it's not sufficient by itself to test an entire application.

(e) Use Cases:

Use cases and scenarios focus on software in its role to enable a human being to do something. Use cases and scenarios tend to resonate with business customers, and if done as part of the requirement process, they sort of magically generate test cases from the requirements.

(f) Code-Based Coverage Models:

Imagine that you have a black-box recorder that writes down every single line of code as it executes. Programmers prefer code coverage. It allows them to attach a number—an actual, hard, real number, such as 75%—to the performance of their unit tests, and they can challenge themselves to improve the score.

- Customer-level coverage tools are expensive, programmer-level tools that tend to assume the team is doing automated unit testing and has a continuous-integration server and a fair bit of discipline.
- After installing the tool, most people tend to focus on statement coverage—the least powerful of the measures.

(g) Regression and High-Volume Test Techniques:

- People spend a lot of money on regression testing, taking the old test ideas described above and rerunning them over and over.
- This is generally done with either expensive users or very expensive programmers spending a lot of time writing and later maintaining those automated tests.

Weaknesses

- Building a record/playback/capture rig for a GUI can be extremely expensive, and it might be difficult to tell whether the application hasn't broken, but has changed in a minor way.

Q.2 Attempt any FOUR the following in brief : [16]

Q.2(a) Describe how to select testing tool. [4]

Ans.: Criteria for Selecting Test Tools:

The Categories for selecting Test Tools are,

(1) Meeting requirements;
(2) Technology expectations;
(3) Training/skills;
(4) Management aspects.

(1) Meeting requirements:

There are plenty of tools available in the market but rarely do they meet all the requirements of a given product or a given organization. Evaluating different tools for different requirements involve significant effort, money, and time. Given of the plethora of choice available, huge delay is involved in selecting and implementing test tools.

(2) Technology expectations:

Test tools in general may not allow test developers to extends/modify the functionality of the framework. So extending the functionality requires going back to the tool vendor and involves additional cost and effort. A good number of test tools require their libraries to be linked with product binaries.

(3) Training/skills:

While test tools require plenty of training, very few vendors provide the training to the required level. Organization level training is needed to deploy the test tools, as the user of the test suite are not only the test team but also the development team and other areas like configuration management.

(4) Management aspects:

A test tool increases the system requirement and requires the hardware and software to be upgraded. This increases the cost of the already- expensive test tool.

OR

Guidelines for selecting a tool:

- The tool must match its intended use. Wrong selection of a tool can lead to problems like lower efficiency and effectiveness of testing may be lost.
- Different phases of a life cycle have different quality-factor requirements. Tools required at each stage may differ significantly.
- Matching a tool with the skills of testers is also essential. If the testers do not have proper training and skill then they may not be able to work effectively.
- Select affordable tools. Cost and benefits of various tools must be compared before making final decision.
- Backdoor entry of tools must be prevented. Unauthorized entry results into failure of tool and creates a negative environment for new tool introduction.

Q.2(b) What is White Box testing? Explain any one technique of static White Box testing. [4]

Ans.: White Box Testing: Classification of Whit e Box.

- This is also known as glass box, clear box, and open box testing.
- In white box testing, test cases are created by looking at the code to detect any Potential failure scenarios.
- The suitable input data for testing various APIs and the special code paths that Need to be tested by analyzing the source code for the application block.
- Therefore, the test plans need to be updated before starting white box testing and only after a stable build of the code is available.
- White box testing assumes that the tester can take a look at the code for the application block and create test cases that look for any potential failure scenarios.
- During white box testing, analyze the code of the application block and prepare test cases for testing the functionality to ensure that the class is behaving in accordance with the specifications and testing for robustness.
- A failure of a white box test may result in a change that requires all black box testing to be repeated and white box testing paths to be reviewed and possibly changed.

Static Testing- Inspections, Structured Walkthroughs, Technical Review:

(1) Inspection

- Inspections are the most formal type of reviews.
- They are highly structured and require training for each participant.
- Inspections are different from peer reviews and walkthroughs in that the person who presents the code, the presenter or reader, isn't the original programmer.
- These forces someone else to learn and understand the material being presented, potentially giving a different slant and interpretation at the inspection meeting.
- The other participants are called inspectors.
- Each is tasked with reviewing the code from a different perspective, such as a User, a tester, or a product support person.
- This helps bring different views of the product under review and very often Identifies different bugs.

- One inspector is even tasked with reviewing the code backward—that is, from the end to the beginning—to make sure that the material is covered evenly And completely.

(2) Walkthrough:

- Walkthroughs are the next step up in formality from peer reviews.
- In a walkthrough, the programmer who wrote the code formally presents (Walks through) it to a small group of five or so other programmers and testers.
- The reviewers should receive copies of the software in advance of the review so they can examine it and write comments and questions that they want to ask at the review.
- Having at least one senior programmer as a reviewer is very important.

(3) Technical Review:

(i) Formal Review:

- A formal review is the process under which static white-box testing is performed.
- A formal review can range from a simple meeting between two programmers to a detailed, rigorous inspection of the code.

There are four essential elements to a formal review

(a) Identify Problems:
(b) Follow Rules:
(c) Prepare:
(d) Write a Report:

(ii) Peer Reviews:

- The easiest way to get team members together and doing their first formal reviews of the software is through peer reviews, the least formal method.
- Sometimes called buddy reviews, this method is really more of a discussion.
- Peer reviews are often held with just the programmer who wrote the code and one or two other programmers or testers acting as reviewers.
- Small group simply reviews the code together and looks for problems and oversights.
- To assure that the review is highly effective all the participants need to make sure that the four key elements of a formal review are in place: Look for problems, follow rules, prepare for the review, and write a report.
- As peer reviews are informal, these elements are often scaled back. Still, just getting together to discuss the code can find bugs.

Structural Testing-Code Functional Testing, Code Coverage Testing, Code Complexity Testing.

Data Flow (Code Functional Testing)

- Data flow coverage involves tracking a piece of data completely through the software.
- At the unit test level this would just be through an individual module or function.
- The same tracking could be done through several integrated modules or even through the entire software product—although it would be more time consuming to do so.
- During data flow, the check is made for the proper declaration of variables declared and the loops used are declared and used properly.

For example

```
1. #include<stdio.h>
2. void main()
3. {
4. int i , fact= 1, n;
5. printf("enter the number ");
6. scanf("%d",&n);
7. for(i =1 ;i <=n;i++)
```

```
8. fact = fact * i;
9. printf ("the factorial of a number is "%d", fact);
10. }
```

Data Coverage (Code Coverage Testing)

- The logical approach is to divide the code just as you did in black-box testing into its data and its states (or program flow).
- By looking at the software from the same perspective, you can more easily map the white-box information you gain to the black-box cases you've already written.
- Consider the data first. Data includes all the variables, constants, arrays, data structures, keyboard and mouse input, files and screen input and output, and I/O to other devices such as modems, networks, and so on.

  For example

```
1. #include<stdio.h>
2. void main()
3. {
4. int i, fact= 1, n;
5. printf("enter the number ");
6. scanf("%d", &n);
7. for(i =1 ;i <=n;i++)
8. fact = fact * i;
9. printf ("the factorial of a number is %d", fact);
10. }
```

The declaration of data is complete with the assignment statement and the variable declaration statements. All the variable declared are properly utilized.

Program Statements and Line Coverage (Code Complexity Testing)

- The most straightforward form of code coverage is called statement coverage or line coverage.
- If you're monitoring statement coverage while you test your software, your goal is to make sure that you execute every statement in the program at least once.
- With line coverage the tester tests the code line by line giving the relevant output.

  For example

```
1. #include<stdio.h>
2. void main()
3. {
4. int i , fact= 1, n;
5. printf("enter the number ");
6. scanf("%d", &n);
7. for(i =1 ;i <=n; i++)
8. fact = fact * i;
9. printf ("the factorial of a number is %d", fact);
10. }
```

Branch Coverage (Code Complexity Testing)

- Attempting to cover all the paths in the software is called path testing.
- The simplest form of path testing is called branch coverage testing.
- To check all the possibilities of the boundary and the sub boundary conditions and it's branching on those values.
- Test coverage criteria requires enough test cases such that each condition in a decision takes on all possible outcomes at least once, and each point of entry to a program or subroutine is invoked at least once.
- Every branch (decision) taken each way, true and false.
- It helps in validating all the branches in the code making sure that no branch leads to abnormal behavior of the application.

Condition Coverage (Code Complexity Testing)

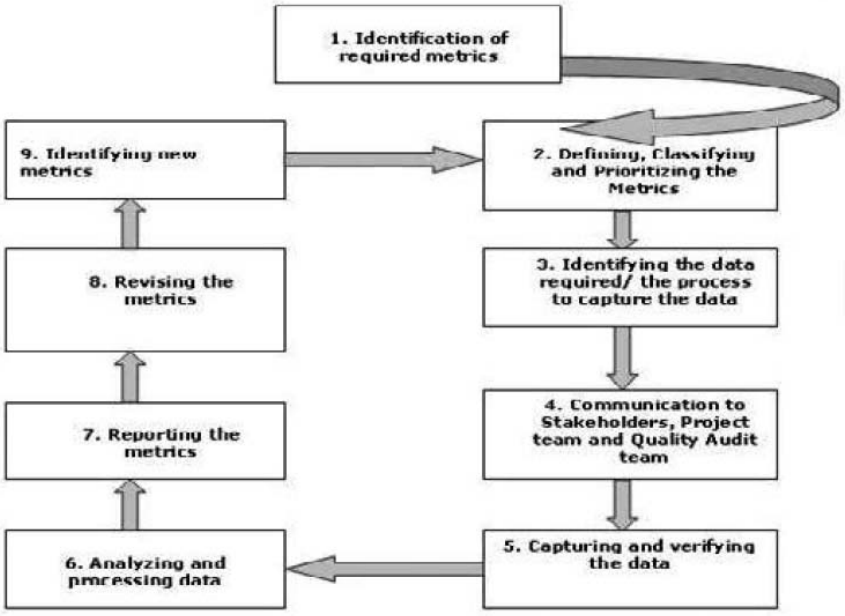
- Just when you thought you had it all figured out, there's yet another Complication to path testing.
- Condition coverage testing takes the extra conditions on the branch statements into account.

Q.2(c) Define Metrics and Measurements. Explain need of Software measurement. [4]

Ans.: A Metric is a measurement of the degree that any attribute belongs to a system, product or process. For example the number of errors per person hours would be a metric. Thus, software measurement gives rise to software metrics. A measurement is an indication of the size, quantity, amount or dimension of a particular attribute of a product or process. For example the number of errors in a system is a measurement.

Software measurement is required to:

- Establish the quality of the current product or process.
- To predict future qualities of the product or process.
- To improve the quality of a product or process.
- To determine the state of the project in relation to budget and schedule.



Q.2(d) Describe equivalence partitioning with example. [4]

Ans.: Equivalence partitioning is a software technique that involves identifying a small set of representative input values that produce as much different output condition as possible. This reduces the number of permutation & combination of input, output values used for testing, thereby increasing the coverage and reducing the effort involved in testing. The set of input values that generate one single expected output is called a partition. When the behavior of the software is the same for a set of values, then the set is termed as equivalence class or partition.

Example: An insurance company that has the following premium rates based on the age group. A life insurance company has base premium of Rs. 500 for all ages. Based on the age group, an additional monthly premium has to pay that is as listed in the table below. For example, a person aged 34 has to pay a premium = Rs. 500 + Rs. 1000 = Rs. 1500

Age group	Extra Premium
Under 35	Rs.1500
35 - 59	Rs.2500
60+	Rs.4000

Based on the equivalence portioning technique, the equivalence partitions that are based on age are given below:

- Below 35 years of age (valid input)
- Between 35 and 59 years of age (valid input)
- Above 6 years of age (valid input)
- Negative age (invalid input)
- Age as 0 (invalid input)
- Age as any three-digit number (valid input)

Q.2(e) How to perform security testing? State elements of security testing. [4]

Ans.: **Security Testing:** Testers must use a risk-based approach, By identifying risks and potential loss associated with those risks in the system and creating tests driven by those risks, the testers can properly focus on areas of code in which an attack is likely to succeed. Therefore risk analysis at the design level can help to identify potential security problems and their impacts. Once identified ranked, software risks can help guide software security. It is a type of non-functional testing. Security testing is basically a type of software testing that's done to check whether the application or the product is secured or not. It checks to see if the application is vulnerable to attacks, if anyone hack the system or login to the application without any authorization. It is a process to determine that an information system protects data and maintains functionality as intended. The security testing is performed to check whether there is any information leakage in the sense by encrypting the application or using wide range of software's and hardware's and firewall etc. Software security is about making software behave in the presence of a malicious attack.

The six basic security concepts / elements that need to be covered by security testing are:

- confidentiality,
- integrity,
- authentication,
- availability,
- authorization and
- Non-repudiation.

Q.3Attempt any FOUR of the following : [16]

Q.3(a) Give any four difference between alpha and beta testing. [4]

Ans.:

Sr. No.	Alpha Testing	Beta Testing
1	Performed at developer's site	Performed at end user's site
2	Performed in controlled environment as developer is present	Performed in uncontrolled environment as developer is not present
3	Less probability of finding of error as it is driven by developers.	High probability of finding of error as end user can use it the way he wants.
4	It is not considered as live application	It is considered as live application
5	It is done during implementation phase of software	It is done as pre-release of software
6	Less time consuming as developer can make necessary changes in given time.	More time consuming. As user has to report bugs if any via appropriate channel.

OR

Sr. No.	Alpha Testing	Beta Testing (Field Testing)
1	It is always performed by the developers at the software development site.	It is always performed by the customers or end users at their own site.
2	Sometimes it is also performed by Independent Testing Team.	It is not performed by Independent Testing Team.

3	Alpha Testing is not open to the market and public	Beta Testing is always open to the market and public.
4	It is conducted for the software application and project.	It is usually conducted for software product.
5	It is always performed in Virtual Environment.	It is performed in Real Time Environment.
6	It is always performed within the organization.	It is always performed outside the organization.
7	Alpha Testing is definitely performed and carried out at the developing organizations location with the involvement of developers.	Beta Testing (field testing) is performed and carried out by users or you can say people at their own locations and site using customer data.
8	It comes under the category of both White Box Testing and Black Box Testing.	It is only a kind of Black Box Testing.
9	Alpha Testing is always performed at the time of Acceptance Testing when developers test the product and project to check whether it meets the user requirements or not.	Beta Testing is always performed at the time when software product and project are marketed.
10	It is always performed at the developer's premises in the absence of the users.	It is always performed at the user's premises in the absence of the development team.
11	Alpha Testing is not known by any other different name.	Beta Testing is also known by the name Field Testing means it is also known as field testing.
12	It is considered as the User Acceptance Testing (UAT) which is done at developer's area.	It is also considered as the User Acceptance Testing (UAT) which is done at customers or users area.

Q.3(b) Explain Defect template. [4]

Ans.: Defect template: A defect report documents an anomaly discovered during testing. It includes all the information needed to reproduce the problem, including the author, release/build number, open/close dates, problem area, problem description, test environment, defect type, how it was detected, who detected it, priority, severity, status, etc.

After uncovering a defect (bug), testers generate a formal defect report. The purpose of a defect report is to state the problem as clearly as possible so that developers can replicate the defect easily and fix it.

Defect Report Template

In most companies, a defect reporting tool is used and the elements of a report can vary. However, in general, a defect report can consist of the following elements.

ID	Unique identifier given to the defect. (Usually Automated)
Project	Project name.
Product	Product name.
Release Version	Release version of the product. (e.g. 1.2.3)
Module	Specific module of the product where the defect was detected.
Detected Build Version	Build version of the product where the defect was detected (e.g. 1.2.3.5)
Summary	Summary of the defect. Keep this clear and concise.

Description	Detailed description of the defect. Describe as much as possible but without Repeating anything or using complex words. Keep it simple but comprehensive.
Steps to Replicate	Step by step description of the way to reproduce the defect. Number the steps.
Actual Result	The actual result you received when you followed the steps.
Expected Results	The expected results.
Attachments	Attach any additional information like screenshots and logs.
Remarks	Any additional comments on the defect.
Defect Severity	Severity of the Defect.
Defect Priority	Priority of the Defect.
Reported By	The name of the person who reported the defect.
Assigned To	The name of the person that is assigned to analyze/fix the defect.
Status	The status of the defect.
Fixed Build Version	Build version of the product where the defect was fixed (e.g. 1.2.3.9)

Q.3(c) With the help of suitable example explain decision table. Why decision table is important? [4]

Ans.: Decision Tables

Conditions	**TC1**	**TC2**	**TC3**	**TC4**
Request login	0	1	1	1
Valid user name entered	X	0	1	1
Valid password entered	X	X	0	1
Actions				
Offer recovery credentials	0	1	1	0
Activate entrybox user name	0	1	1	0
Activate entrybox password	0	0	1	0
Enter priviliged area	0	0	0	1

- Decision table testing is black box test design technique to determine the test scenarios for complex business logic.
- Decision tables provide a systematic way of stating complex business rules, which is useful for developers as well as for testers.
- Decision tables can be used in test design whether or not they are used in specifications, as they help testers explore the effects of combinations of different inputs and other software states that must correctly implement business rules.
- It helps the developers to do a better job can also lead to better relationships with them.
- Testing combinations can be a challenge, as the number of combinations can often be huge.
- Testing all combinations may be impractical if not impossible.
- We have to be satisfied with testing just a small subset of combinations but making the choice of which combinations to test and which to leave out is also important.
- If you do not have a systematic way of selecting combinations, an arbitrary subset will be used and this may well result in an ineffective test effort.

Importance of Decision Table:

Essentially it is a structured exercise to formulate requirements when dealing with complex business rules. Decision tables are used to model complicated logic. They can make it easy to see that all possible combinations of conditions have been considered and when conditions are missed, it is easy to see.

Q.3(d) How to identify resource requirement of test plan? [4]

Ans.: Resource requirement is a detailed summary of all types of resources required to complete project task. Resource could be human, equipment and materials needed to complete a project. The resource requirement and planning is important factor of the test planning because helps in determining the number of resources (employee, equipment…) to be used for the project. Therefore, the Test Manager can make the correct schedule & estimation for the project.

Some of the following factors need to be considered:

- Machine configuration (RAM, processor, disk) needed to run the product under test.
- Overheads required by test automation tools, if any
- Supporting tools such as compilers, test data generators, configuration management tools.
- The different configurations of the supporting software (e.g. OS)that must be present

Sr. No.	Member	Tasks
1	Test Manager	Manage the whole project Define project directions Acquire appropriate resources
2	Tester	Identifying and describing appropriate test techniques/tools/automation architecture Verify and assess the Test Approach Execute the tests, Log results, Report the defects. Tester could be in-sourced or out-sourced members base on the project budget For the task which required low skill, I recommend you choose outsourced members to save project cost.
3	Developer in Test	Implement the test cases, test program, test suite etc.
4	Test Administrator	Builds up and ensures test environment and assets are managed and maintained Support Tester to use the test environment for test execution
5	SQA members	Take in charge of quality assurance Check to confirm whether the testing process is meeting specified requirements

- Special requirements for running machine-intensive tests such as load tests and performance tests.
- Appropriate number of licenses of all the software

OR

Human Resource:

The following table represents various members in your project team

System Resource:

For testing, a web application, you should plan the resources as following tables:

Sr. No.	Resources	Descriptions
1	Server	Install the web application under test This includes a separate web server, database server, and application serv if applicable
2	Test tool	The testing tool is to automate the testing, simulate the user operation, generate the test results There are tons of test tools you can use for this project such as Selenium QTP…etc.

3	Network	You need a Network include LAN and Internet to simulate the real business and user environment
4	Computer	The PC which users often use to connect the web server

Q.3(e) Illustrate process of bi-directional integration testing. State its two advantages and disadvantages. [4]

Ans.:

1. Bi-directional Integration, is a kind of integration testing process that combines top-down and bottom-up testing.
2. With an experience in delivering Bi-directional testing projects custom software development services provide the best quality of the deliverables right from the development of software process.
3. Bi-directional Integration testing is a vertical incremental testing strategy that tests the bottom layers and top layers and tests the integrated system in the computer software development process.
4. Using stubs, it tests the user interface in isolation as well as tests the very lowest level functions using drivers.
5. Bi-directional Integration testing combines bottom-up and top-down testing.
6. Bottom-up testing is a process where lower level modules are integrated and then tested.
7. This process is repeated until the component of the top of the hierarchy is analyzed. It helps custom software development services find bugs easily without any problems.
8. Top down testing is a process where the top integrated modules are tested and the procedure is continued till the end of the related module.
9. Top down testing helps developers find the missing branch link easily.

OR

Process of Bidirectional testing:

- Bottom up testing starts from middle layer and goes upward to the top layer. For a very big system, bottom up approach starts at a subsystem level and goes upwards.
- Top down testing starts from the middle layer and goes downward. For a very big system, top down approach, starts at subsystem level and goes downwards.
- Big band approach is followed for middle layer. From this layer, bottom up approach goes upwards and top down approach goes downwards.

Advantages:

- This approach is useful is useful for very large projects having several projects. When development follows a spiral model and module itself is as large as a system.
- Both top down and bottom up approach starts at the start of the schedule.
- It needs more resources and big teams for performing both, methods of testing at a time or one after the other.

Disadvantages:

- It represents very high cost of testing as lot of testing is done.
- It cannot be used for smaller systems with huge interdependence between different modules.
- Different skill tests are required for testers at different level as modules are separate systems handling separate domains.

Q.4(a) Attempt any THREE of the following: [12]

Q.4(a) (i) Explain test deliverables in details. [4]

Ans.: Test deliverables

The test plan also identifies the deliverables that should come out of the test cycle/testing activity. The Deliverables includes the following, all reviewed and approved by the appropriate people.

(i) The test plan itself master test plan and previous other test plans for the project)
(ii) Test case design specifications.
(iii) Test cases, including any automation that is specified in the plan.
(iv) Test logs produced by running the tests.
(v) Test summary reports

Q.4(a) (ii) What is automated testing? Write down advantages of using automated testing tools in software testing. [4]

Ans.:
- An automated testing tool is able to playback pre-recorded and predefined actions, Compare the results to the expected behavior and report the success or failure of these manual tests to a test engineer.
- Once automated tests are created they can easily be repeated and they can be extended to perform tasks impossible with manual testing.
- Because of this, savvy managers have found that automated software testing is an essential component of successful development projects.

Benefits of automation testing:

(i) Speed: Think about how long it would take you to manually try a few thousand test cases for the windows Calculator. You might average a test case every five seconds or so. Automation might be able to run 10, 100 even 1000 times that fast.

(ii) Efficiency: While you are busy running test cases, you can't be doing anything else. If you have a test tool that reduces the time it takes for you to run your tests, you have more time for test planning and thinking up new tests.

(iii) Accuracy and Precision: After trying a few hundred cases, your attention may reduce and you will start to make mistakes .A test tool will perform the same test and check the result perfectly, each and every time.

(iv) Resource Reduction: Sometimes it can be physically impossible to perform a certain test case. The number of people or the amount of equipment required to create the test condition could be prohibitive. A test tool can used to simulate the real world and greatly reduce the physical resources necessary to perform the testing.

(v) Simulation and Emulation: Test tools are used to replace hardware or software that would normally interface to your product. This "face" device or application can then be used to drive or respond to your software in ways that you choose-and ways that might otherwise be difficult to achieve.

(vi)Relentlessness: Test tool and automation never tire or give up. It will continuously test the software.

OR

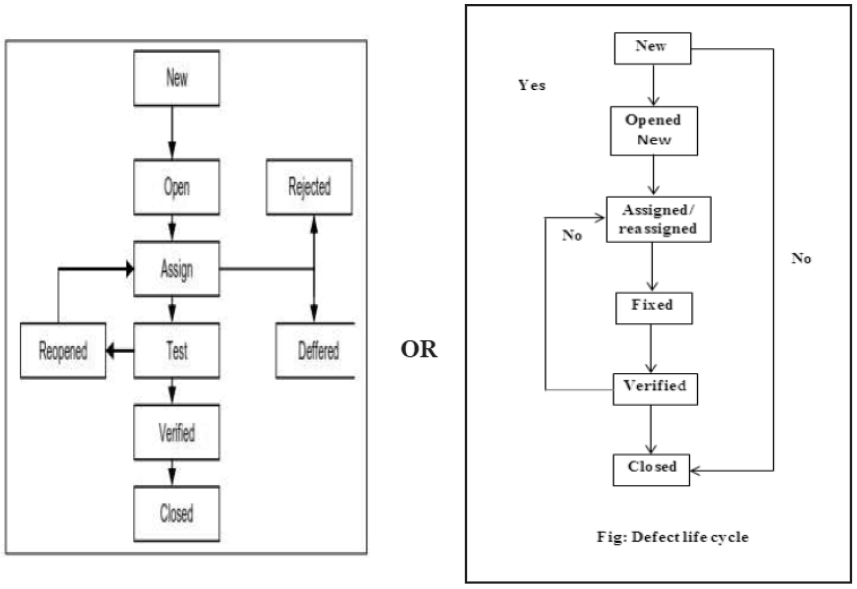
Benefits of Automation Testing are:

1. **Save Time /Speed:** Due to advanced computing facilities, automation test tools prevail in speed of processing the tests. Automation saves time as software can execute test cases faster than human.
2. **Reduces the tester's involvement in executing tests:** It relieves the testers to do some other work.
3. **Repeatability/Consistency:** The same tests can be re-run in exactly the same manner eliminating the risk of human errors such as testers forgetting their exact actions, intentionally omitting steps from the test scripts, missing out steps from the test script, all of which can result in either defects not being identified or the reporting of invalid bugs (which can again, be time consuming for both developers and testers to reproduce)
4. **Simulated Testing:** Automated tools can create many concurrent virtual users/data and effectively test the project in the test environment before releasing the product.

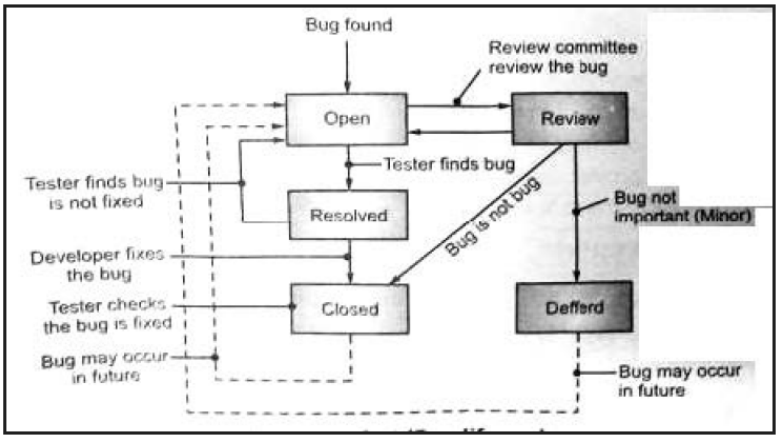
5. **Test case design:** Automated tools can be used to design test cases also. Through automation, better coverage can be guaranteed than if done manually.
6. **Reusable:** The automated tests can be reused on different versions of the software, even if the interface changes.
7. **Avoids human mistakes:** Manually executing the test cases may incorporate errors. But this can be avoided in automation testing.
8. **Internal Testing:** Testing may require testing for memory leakage or checking the coverage of testing. Automation can done this easily.
9. **Cost Reduction:** If testing time increases cost of the software also increases. Due to testing tools time and therefore cost is reduced.

Q.4(a) (iii) Explain defect life cycle to identify status of defect with proper labelled diagram. [4]

Ans.:



OR



The different states of bug life cycle are as shown in the above diagram:

- **New:** When the bug is posted for the first time, its state will be "NEW". This means that the bug is not yet approved.
- **Open:** After a tester has posted a bug, the lead of the tester approves that the bug is genuine and he changes the state as "OPEN".
- **Assign:** Once the lead changes the state as "OPEN", he assigns the bug to corresponding developer or developer team. The state of the bug now is changed to "ASSIGN".
- **Test/Retest:** Once the developer fixes the bug, he has to assign the bug to the testing team for next round of testing. Before he releases the software with bug fixed, he changes the state of bug to "TEST". It specifies that the bug has been fixed and is

released to testing team.// At this stage the tester do the retesting of the changed code which developer has given to him to check whether the defect got fixed or not.

- **Deferred:** The bug, changed to deferred state means the bug is expected to be fixed in next releases. The reasons for changing the bug to this state have many factors. Some of them are priority of the bug may be low, lack of time for the release or the bug may not have major effect on the software.
- **Rejected:** If the developer feels that the bug is not genuine, he rejects the bug. Then the state of the bug is changed to "REJECTED".
- **Verified:** Once the bug is fixed and the status is changed to "TEST", the tester tests the bug. If the bug is not present in the software, he approves that the bug is fixed and changes the status to "VERIFIED".
- **Reopened:** If the bug still exists even after the bug is fixed by the developer, the tester changes the status to "REOPENED". The bug traverses the life cycle once again.
- **Closed:** Once the bug is fixed, it is tested by the tester. If the tester feels that the bug no longer exists in the software, he changes the status of the bug to "CLOSED". This state means that the bug is fixed, tested and approved.
- **Fixed:** When developer makes necessary code changes and verifies the changes then he/she can make bug status as „Fixed" and the bug is passed to testing team.
- **Pending retest:** After fixing the defect the developer has given that particular code for retesting to the tester. Here the testing is pending on the testers end. Hence its status is pending retest.

Optional

- **Duplicate:** If the bug is repeated twice or the two bugs mention the same concept of the bug, then one bug status is changed to "duplicate".
- **Not a bug:** The state given as "Not a bug" if there is no change in the functionality of the application. For an example: If customer asks for some change in the look and field of the application like change of color of some text then it is not a bug but just some change in the looks of the application.

Q.4(a) (iv) How to prepare test plan? [4]

Ans.: Steps to prepare test plan:

Like any project, the testing also should be driven by a plan. The test plan acts as the anchor for the execution, tracking and reporting of the entire testing project. Activities of test plan:

1. Scope Management: Deciding what features to be tested and not to be tested.
2. Deciding Test approach /strategy: Which type of testing shall be done like configuration, integration, localization etc.
3. Setting up criteria for testing: There must be clear entry and exit criteria for different phases of testing. The test strategies for the various features and combinations determined how these features and combinations would be tested.
4. Identifying responsibilities, staffing and training needs.
5. Identifying resource requirements
6. Identifying test deliverables.
7. Testing tasks: size and effort estimation.

OR

1. Test Plan is a document that is the point of reference based on which testing is carried out within the QA team.
2. Test plan is not static and is updated on an on demand basis.
3. The more detailed and comprehensive the Test plan, the more successful the testing activity.
4. The test plan keeps track of possible tests that will be run on the system after coding.

5. The test plan is a document that develops as the project is being developed.
6. Record tests as they come up
7. Test error prone parts of software development.
8. The initial test plan is abstract and the final test plan is concrete.
9. The initial test plan contains high level ideas about testing the system without getting into the details of exact test cases.
10. The most important test cases come from the requirements of the system.
11. When the system is in the design stage, the initial tests can be refined a little.
12. During the detailed design or coding phase, exact test cases start to materialize.
13. After coding, the test points are all identified and the entire test plan is exercised on the software.

Q.4(b) Attempt any ONE of the following: [6]

Q.4(b) (i) Describe V-model with labelled diagram. State its any two advantages and disadvantages. Also write where it is applicable. [6]

Ans.: V-model means verification and validation model. It is sequential path of execution of processes. Each phase must be completed before the next phase begins. Under V-model, the corresponding testing phase of the development phase is planned in parallel. So there is verification on one side of V & validation phase on the other side of V.

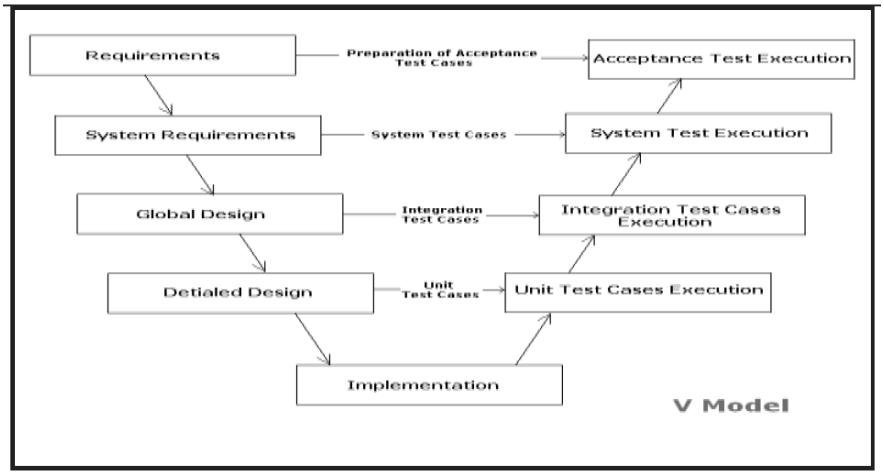
Verification Phase:

1. **Overall Business Requirement:** In this first phase of the development cycle, the product requirements are understood from customer perspective. This phase involves detailed communication with the customer to understand his expectations and exact requirements. The acceptance test design planning is done at this stage as business requirements can be used as an input for acceptance testing.
2. **Software Requirement:** Once the product requirements are clearly known, the system can be designed. The system design comprises of understanding & detailing the complete hardware, software & communication set up for the product under development. System test plan is designed based on system design. Doing this at earlier stage leaves more time for actual test execution later.
3. **High level design:** High level specification are understood & designed in this phase. Usually more than one technical approach is proposed & based on the technical & financial feasibility, the final decision is taken. System design is broken down further into modules taking up different functionality.
4. **Low level design:** In this phase the detailed integral design for all the system modules is specified. It is important that the design is compatible with the other modules in the system & other external system. Components tests can be designed at this stage based on the internal module design
5. **Coding:** The actual coding of the system modules designed in the design phase is taken up in the coding phase. The base suitable programming language is decided base on requirements. Coding is done based on the coding guidelines & standards.

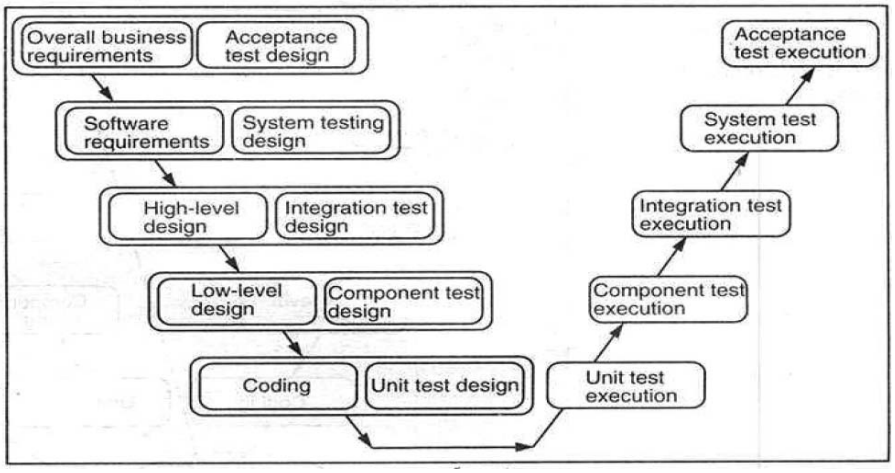
Validation:

- **Unit Testing:** Unit testing designed in coding are executed on the code during this validation phase. This helps to eliminate bugs at an early stage.
- **Components testing:** This is associated with module design helps to eliminate defects in individual modules.
- **Integration Testing:** It is associated with high level design phase & it is performed to test the coexistence & communication of the internal modules within the system
- **System Testing:** It is associated with system design phase. It checks the entire system functionality & the communication of the system under development with external systems. Most of the software & hardware compatibility issues can be uncovered using system test execution.

- **Acceptance Testing:** It is associated with overall & involves testing the product in user environment. These tests uncover the compatibility issues with the other systems available in the user environment. It also uncovers the nonfunctional issues such as load & performance defects in the actual user environment.



OR



Advantages of V-model:	Disadvantages of V-model:
• Simple and easy to use. • Testing activities like planning, test designing happens well before coding. • Saves a lot of time. • Higher chance of success over the waterfall model. • Proactive defect tracking – that is defects are found at early stage. • Avoids the downward flow of the defects. • Works well for small projects where requirements are easily understood.	• Very rigid and least flexible. • Software is developed during the implementation phase, so no early prototypes of the software are produced. • If any changes happen in midway, then the test documents along with requirement documents has to be updated.

When to use the V-model:

- The V-shaped model should be used for small to medium sized projects where requirements are clearly defined and fixed.
- The V-Shaped model should be chosen when ample technical resources are available with needed technical expertise.

Q.4(b) (ii) With the help of example explain Boundary Value Analysis. [6]

Ans.: Most of the defects in software products hover around conditions and boundaries. By conditions, we mean situations wherein, based on the values of various variables, certain actions would have to be taken. By boundaries, we mean "limits" of values of the various variables.

- This is one of the software testing technique in which the test cases are designed to include values at the boundary. If the input data is used within the boundary value limits, then it is said to be Positive Testing. If the input data is picked outside the boundary value limits, then it is said to be Negative Testing.
- Boundary value analysis is another black box test design technique and it is used to find the errors at boundaries of input domain rather than finding those errors in the center of input.
- Each boundary has a valid boundary value and an invalid boundary value. Test cases are designed based on the both valid and invalid boundary values. Typically, we choose one test case from each boundary.
- Same examples of Boundary value analysis concept are:

One test case for exact boundary values of input domains each means 1 and 100. One test case for just below boundary value of input domains each means 0 and 99. One test case for just above boundary values of input domains each means 2 and 101.

For Example: A system can accept the numbers from 1 to 10 numeric values. All other numbers are invalid values. Under this technique, boundary values 0, 1, 2, 9,10,11 can be tested.

Another Example is in exam has a pass boundary at 40 percent, merit at 75 percent and Distinction at 85 percent. The Valid Boundary values for this scenario will be as follows:

49, 50 - for pass

74, 75 - for merit

84, 85 - for distinction

Boundary values are validated against both the valid boundaries and invalid boundaries. The Invalid Boundary Cases for the above example can be given as follows

0 - for lower limit boundary value

101 - for upper limit boundary value

- Boundary value analysis is a black box testing and is also applies to white box testing. Internal data structures like arrays, stacks and queues need to be checked for boundary or limit conditions; when there are linked lists used as internal structures, the behavior of the list at the beginning and end have to be tested thoroughly.
- Boundary value analysis help identify the test cases that are most likely to uncover defects.

Q.5 Attempt any TWO of the following: [16]

Q.5(a) Explain following concepts related to Integration testing with neat & labelled diagram : [8]

(i) Top-down testing (ii) Bottom-up testing

Ans.:

- Integration is process by which components are aggregated to create larger components.
- Testing the data flow or interface between two features is known as integration testing.
- It mainly focuses on I/O protocols, parameters passing between different unit's modules and/or system etc.

Types of Integration testing:

1. Top-down Testing:

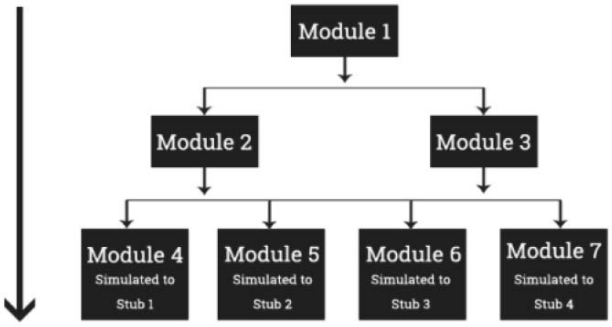
In this approach testing is conducted from main module to sub module. If the sub module is not developed a temporary program called STUB is used for simulate the sub module.

Advantages:

- Advantageous if major flaws occur toward the top of the program.
- Once the I/O functions are added, representation of test cases is easier.
- Early skeletal Program allows demonstrations and boosts morale.

Disadvantages:

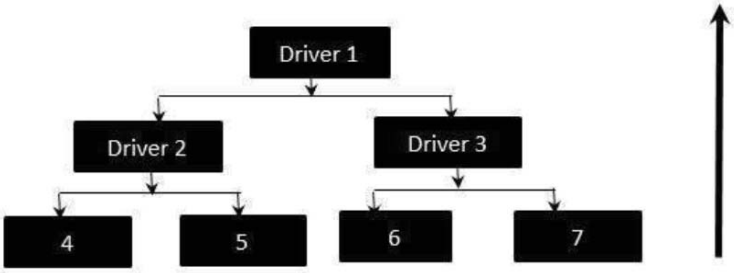
- Stub modules must be produced
- Stub Modules are often more complicated than they first appear to be.
- Before the I/O functions are added, representation of test cases in stubs can be difficult.
- Test conditions may be impossible, or very difficult, to create.
- Observation of test output is more difficult.
- Allows one to think that design and testing can be overlapped.
- Induces one to defer completion of the testing of certain modules.



2. **Bottom-up testing:**
   In this approach testing is conducted from sub module to main module, if the main module is not developed a temporary program called DRIVERS is used to simulate the main module.

Advantages:

- Advantageous if major flaws occur toward the bottom of the program.
- Test conditions are easier to create.
- Observation of test results is easier.
- Driver Modules must be produced.
- The program as an entity does not exist until the last module is added.



Q.5(b) Prepare any eight test cases for admission form of college admission. [8]

Ans.: Consider the college admission form having different fields such as Student's Name, Father's Name, Zip code, Address, Phone, Caste, admission type, S.S.C percentage, SC Board, Submit button, Reset button.

Test Case Id	Test case Objectives	Input Data	Expected Result	Actual Result	Status
TC1	Name field	Any name (abcd xyz)	It should accept the name	The name is accepted	Pass

TC2	Phone Field	Any number Having less than 10 digits(1234)	It should not Accept. Should give error message "Please enter valid phone number"	Error message "Please enter valid phone number"	Pass
TC3	Phone Field	Any Alphabets (abcde)	It should give error message as "Only Numbers"	Error message as "Only Numbers"	Pass
TC4	SSC Percentage Field	65	It should accept	It accepted	Pass
TC5	SSC Percentage Field	30	It should not accepted should give error message	Gives error message	Pass
TC6	Address field	Any characters (A-51, Market road, Mumbai)	It should accept	It accepted	Pass
TC7	Zip code	Any six digit number(431401)	It should accept the number	It accepted	Pass
TC8	Required fields should be filled in the form	Data in the mandatory fields.	If it is filled.	Do not display Error Message	Pass

Q.5(c) Write a short note on : **[8]**

(i) Load testing **(ii) Stress testing**

(iii) Recovery testing **(iv) Usability testing**

Ans.: **(i) Load testing:** Load is testing the software under customer expected load. In order to perform load testing on the software you feed it all that it can handle. Operate the software with largest possible data files. If the software operates on peripherals such as printer, or communication ports, connect as many as you can. If you are testing an internet server that can handle thousands of simultaneous connections, do it. With most software it is important for it to run over long periods. Some software's should be able to run forever without being restarted. So Time acts as a important variable.

(ii) Stress testing: Stress testing is testing the software under less than ideal conditions. So subject your software to low memory, low disk space, slow cpus, and slow modems and so on. Look at your software and determine what external resources and dependencies it has. Stress testing is simply limiting them to bare minimum. With stress testing you

starve the software. For e.g. Word processor software running on your computer with all available memory and disk space, it works fine. But if the system runs low on resources you had a greater potential to expect a bug. Setting the values to zero or near zero will make the software execute different path as it attempt to handle the tight constraint. Ideally the software would run without crashing or losing data.

(iii) Recovery testing: Recovery testing is a type of non-functional testing. Recovery testing is done in order to check how fast and better the application can recover after it has gone through any type of crash or hardware failure etc.

- Recovery testing is the forced failure of the software in a variety of ways to verify that recovery is properly performed.
- Determining the feasibility of the recovery process.
- Verification of the backup facilities.
- Ensuring proper steps are documented to verify the compatibility of backup facilities.
- Providing Training within the team.
- Demonstrating the ability of the organization to recover from all critical failures.
- Maintaining and updating the recovery plan at regular intervals.

(iv) Usability testing: Usability testing, a non-functional testing technique that is a measure of how easily the system can be used by end users.It is difficult to evaluate and measure but can be evaluated based on the below parameters:

- Levels of Skill required learn/use the software. It should maintain the balance for both novice and expert user.
- Time required to get used to in using the software.
- The measure of increase in user productivity if any.
- Assessment of a user's attitude towards using the software.
- Usability testing, a non-functional testing technique that is a measure of how easily the system can be used by end users.
- It is difficult to evaluate and measure but can be evaluated based on the below parameters.
- Levels of Skill required learn/use the software. It should maintain the balance for both novice and expert user.
- Time required to get used to in using the software.
- The measure of increase in user productivity if any.

Q.6 Attempt any FOUR of the following: [16]

Q.6(a) Describe compatibility testing with an example. [4]

Ans.: (i) Compatibility Testing is a type of Software testing to check whether your software is capable of running on different hardware, operating systems, applications, network environments.

(ii) It is a type of the Non-functional testing.

(iii) There are two types of compatibility testing:

- Forward Compatibility.
- Backward Compatibility.

Forward Compatible:

If something is forward compatible, it will work with future versions of the software.

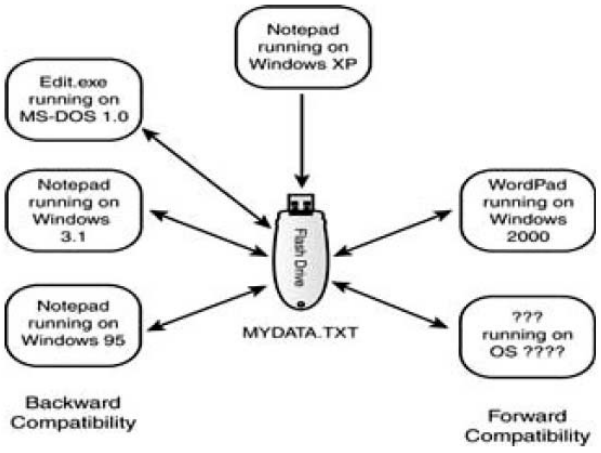
Backward Compatible:

If something is backward compatible, it will work with previous versions of the software.

E.g.

- Cutting text from a web page and pasting it into document opened in your word processor.

- Saving accounting data from one spreadsheet program and then loading into a completely different spreadsheet program.



Q.6(b) Which parameters are considered while writing good defect report? Also write contents of defect template. **[4]**

Ans.: A defect report documents an anomaly discovered during testing. It includes all the information needed to reproduce the problem, including the author, release/build number, open/close dates, problem area, problem description, test environment, defect type, how it was detected, who detected it, priority, severity, status, etc. After uncovering a defect (bug), testers generate a formal defect report. The purpose of a defect report is to state the problem as clearly as possible so that developers can replicate the defect easily and fix it.

DEFECT REPORT TEMPLATE: In most companies, a defect reporting tool is used and the elements of a report can vary. However, in general, a defect report can consist of the following elements.

ID	Unique identifier given to the defect. (Usually Automated)
Project	Project name.
Product	Product name.
Release Version	Release version of the product. (e.g. 1.2.3)
Module	Specific module of the product where the defect was detected.
Detected Build Version	Build version of the product where the defect was detected (e.g. 1.2.3.5)
Summary	Summary of the defect. Keep this clear and concise.
Description	Detailed description of the defect. Describe as much as possible but without Repeating anything or using complex words. Keep it simple but comprehensive.
Steps to Replicate	Step by step description of the way to reproduce the defect. Number the steps.
Actual Result	The actual result you received when you followed the steps.
Expected Results	The expected results.
Attachments	Attach any additional information like screenshots and logs.
Remarks	Any additional comments on the defect.
Defect Severity	Severity of the Defect.
Defect Priority	Priority of the Defect.
Reported By	The name of the person who reported the defect.
Assigned To	The name of the person that is assigned to analyze/fix the defect.
Status	The status of the defect.
Fixed Build Version	Build version of the product where the defect was fixed (e.g. 1.2.3.9)

Q.6(c) What is test planning and test management? [4]

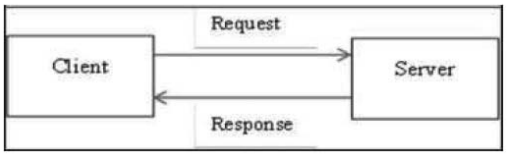
Ans.: **Test Planning:** Like any project, the testing also should be driven by a plan. The test plan acts as the anchor for the execution, tracking and reporting of the entire testing project. Activities of test plan:

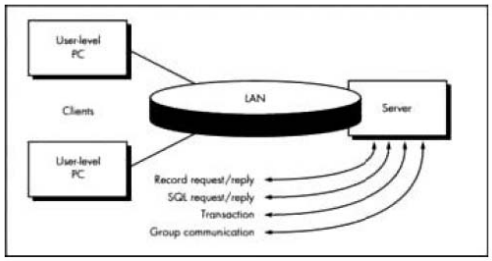
1. **Scope Management:** Deciding what features to be tested and not to be tested.
2. **Deciding Test approach /strategy:** Which type of testing shall be done like configuration, integration, localization etc.
3. **Setting up criteria for testing:** There must be clear entry and exit criteria for different phases of testing. The test strategies for the various features and combinations determined how these features and combinations would be tested.
4. Identifying responsibilities, staffing and training needs

Test Management: It concerned with both test resource and test environment management. It is the role of test management to ensure that new or modified service products meet business requirements for which they have been developed or enhanced.

Q.6(d) With the help of diagram describe client-server testing. [4]

Ans.:





In Client-server testing there are several clients communicating with the server.

1. Multiple users can access the system at a time and they can communicate with the server.
2. Configuration of client is known to the server with certainty.
3. Client and server are connected by real connection.
4. Testing approaches of client server system.

1. **Component Testing:** One need to define the approach and test plan for testing client and server individually. When server is tested there is need of a client simulator, whereas testing client a server simulator, and to test network both simulators are used at a time.

2. **Integration testing:** After successful testing of server, client and network, they are brought together to form system testing.

3. **Performance testing:** System performance is tested when number of clients are communicating with server at a time. Volume testing and stress testing may be used for testing, to test under maximum load as well as normal load expected. Various interactions may be used for stress testing.

4. **Concurrency Testing:** It is very important testing for client-server architecture. It may be possible that multiple users may be accessing same record at a time, and concurrency testing is required to understand the behavior of a system in this situation.

5. **Disaster Recovery/ Business continuity testing:** When the client server are communicating with each other , there exit a possibility of breaking of the communication due to various reasons or failure of either client or server or link

connecting them. The requirement specifications must describe the possible expectations in case of any failure.

6. **Testing for extended periods:** In case of client server applications generally server is never shutdown unless there is some agreed Service Level Agreement (SLA) where server may be shut down for maintenance. It may be expected that server is running 24X7 for extended period. One needs to conduct testing over an extended period to understand if service level of network and server deteriorates over time due to some reasons like memory leakage.

7. **Compatibility Testing:** Client server may be put in different environments when the users are using them in production. Servers may be in different hardware, software, or operating system environment than the recommended. Other testing such as security testing and compliance testing may be involved if needed, as per testing and type of system.

Q.6(e) What are types of test report? Write contents of test summary report. [4]

Ans.: Test reporting is a means of achieving communication through the testing cycle. There are 3 types of test reporting.

1. **Test incident report:** A test incident report is communication that happens through the testing cycle as and when Defects are encountered. A test incident report is an entry made in the defect repository each defect has a unique id to identify incident. The high impact test incident are Highlighted in the test summary report.

2. **Test cycle report:** A test cycle entails planning and running certain test in cycle, each cycle using a different build of the product .As the product progresses through the various cycles it is expected to stabilize.

   Test cycle report gives :
   - A summary of the activities carried out during that cycle.
   - Defects that are uncovered during that cycle based on severity and impact
   - Progress from the previous cycle to the current cycle in terms of defect fixed
   - Outstanding defects that not yet to be fixed in cycle
   - Any variation observed in effort or schedule

3. **Test summary report:** The final step in a test cycle is to recommend the suitability of a product for release. A report that summarizes the result of a test cycle is the test summary report. There are two types of test summary report:
   - Phase wise test summary ,which is produced at the end of every phase
   - Final test summary report.

   A Summary report should content:
   - Test Summary report Identifier
   - Description : Identify the test items being reported in this report with test id
   - Variances: Mention any deviation from test plans, test procedures, if any.

4. **Summary of results:** All the results are mentioned here with the resolved incidents and their solutions.

5. Comprehensive assessment and recommendation for release should include Fit for release assessment and recommendation of release.

Q.6(f) List all defect classification. Also describe any one defect in brief. [4]

Ans.: List of Defect Classification:

1. Requirements and specification defect
2. Design Defects
3. Coding Defects
4. Testing Defect

Requirements and specification defect: Requirement related defects arise in a product when one fails to understand what is required by the customer. These defects may be due to customer gap, where the customer is unable to define his requirements, or producer gap, where developing team is not able to make a product as per requirements. Defects injected in early phases can persist and be very difficult to remove in later phases. Since any requirements documents are written using natural language representation, there are very often occurrences of ambiguous, contradictory, unclear, redundant and imprecise requirements. Specifications are also developed using natural language representations.

Design Defects: Design defects occur when system components, interactions between system components, interactions between the outside software/hardware, or users are incorrectly designed. This covers in the design of algorithms, control, logic/ data elements, module interface descriptions and external software/hardware/user interface descriptions. Design defects generally refer to the way of design creation or its usage while creating a product. The customer may or may not be in a position to understand these defects, if structures are not correct. They may be due to problems with design creation and implementation during software development life cycle.

Coding Defects: Coding defects may arise when designs are implemented wrongly. If there is absence of development/coding standards or if they are wrong, it may lead to coding defects. Coding defects are derived from errors in implementing the code. Coding defect classes are closely related to design defect classes especially if pseudo code has been used for detailed design. Some coding defects come from a failure to understand programming language constructs, and miscommunication with the designers. Others may have transcription or omission origins. At times it may be difficult to classify a defect as a design or as a coding detect.

Testing Defect: Testing defect are defects introduced in an application due to wrong testing, or defects in the test artifact leading to wrong testing. Defects which cannot be reproduced, or are not supported by requirement or are duplicate may represent a false call. In this defects includes

1. **Test-design defect:** test-design defect refers to defects in test artifacts. There can be defects in test plans, test scenarios, test cases and test data definition which can lead to defect in software.
2. **Test-environment defect:** this defect may arise when test environment is not set properly. Test environment may be comprised of hardware, software, simulator and people doing testing.
3. **Test-tool defects:** any defects introduced by a test tool may be very difficult to find and resolve, as one may have to find the defect using manual test as against automated tools.

OR

Software Defects/ Bugs are normally classified as per:

- Severity / Impact
- Probability / Visibility
- Priority / Urgency
- Related Dimension of Quality
- Related Module / Component
- Phase Detected
- Phase Injected

❑ ❑ ❑ ❑ ❑